Kampf den Spaghetti — strukturiertes Programmieren (Teil 1)

Das ist keine Propaganda gegen italienische Teigwaren! Es soll vielmehr einer weit verbreiteten Unart zu Leibe gerückt werden, dem Einfach-drauflos-Tippen beim Programmieren, ohne rechte Idee, was das Programm eigentlich können soll.

Es gibt immer noch viele Hobby-Programmierer, die sich beim Anblick einer Tastatur nicht mehr halten können und einfach drauflos tippen. Besonders bei Anfängern ist dies weit verbreitet (das Ausprobieren einzelner Befehle bildet selbstverständlich eine Ausnahme). Hauptsache, das Endprodukt der Hackerei (das Programm) funktioniert hinterher, oder? Weit gefehlt! Denn was nützt Ihnen ein Programm, bei dem Sie nach einem halben Jahr selbst nicht mehr durchblicken und es neu schreiben müssen?

Das kann Ihnen niemals passieren? Besitzen Sie Ihren Computer denn schon ein halbes Jahr? Nun, Spaß beiseite. Wenn man bei Programmen auch nach einer längeren Zeitspanne noch Änderungen einbauen möchte, etwa um es an neu erworbene Peripherie anzupassen oder das Programm eines anderen mit den eigenen Gegebenheiten abzustimmen, hat man es um so leichter, je eindeutiger und strukturierter das Programm aufgebaut ist. Außerdem verbessert es die Qualität des späteren Programms beträchtlich. Besonders für Anfänger ist es ratsam, sich gleich zu Beginn anzugewöhnen, strukturiert zu programmieren, denn eine einmal erlernte Unart kann man nur schwer wieder ablegen. Wozu sich also quälen?

In der Computer-Fachliteratur ist einiges über die verschiedenen Darstellungsmöglichkeiten von »Struktogrammen« zu lesen. Eine Anleitung jedoch, wie mit dem Werkzeug »Struktogramm« in der Praxis umzugehen ist, eine Art »Kochrezept«, fehlt meist. Diesem Mangel soll hier abgeholfen werden. Es sollen nämlich nicht nur die Darstellungssysteme klarer, sondern auch der Umgang damit erleichtert werden.

Strukturiertes Programmieren ist im Prinzip unabhängig von der Programmiersprache. Es gibt selbstverständlich Sprachen, die das strukturierte Programmieren besonders unterstützen (zum Beispiel Pascal), genau wie es auch solche gibt, die es beinahe unmöglich machen (beispielsweise sehr maschinennahe Sprachen wie Assembler). Einige behaupten zwar, in Basic sei strukturiertes Programmieren ein Unding, das soll aber nicht abschrecken, es auch in dieser Sprache zu versuchen; schon allein deshalb, weil unzählige Heimcomputer vom Hersteller mit Basic ausgerüstet sind.

Basic mit Struktur

Eigentlich sollte jedes Programm eine gewisse »Grundstruktur« aufweisen. Es kann vorkommen, daß einige Elemente dieser Grundstruktur sehr klein ausfallen, oder sogar ganz wegfallen. Aber im Prinzip hat jedes Programm (vom Spielprogramm bis hin zum Finanzbuchhaltungs-Programm) folgende Grundstruktur:

— Vorbereiten zur Datenaufnahme. (Felder dimensionieren, Konstanten definieren, Speicher löschen oder mit bestimmten Bitmustern füllen und so weiter)

— Daten entgegennehmen. (Sowohl durch Einlesen aus dem Programm, zum Beispiel DATA-Zeilen, oder durch manuell eingetastete Werte von dem Operator, oder durch Übernahme von Tape/Floppy und so weiter)

— Daten verarbeiten — Programmausführung (Jetzt erst wird der eigentliche Algorithmus beziehungsweise die Rechen- oder Datenverarbeitungs-Vorschrift abgearbeitet.)

— Ergebnisse und Daten ausgeben. (Die Form, also Grafik, als Tabelle, als Zeichnung und so weiter ist dabei egal.)

Das Wichtigste beim Programmieren und beim strukturierten Programmieren insbesondere ist nicht etwa die Arbeit an der Tastatur, sondern die vorherige Erarbeitung des Programms am Schreibtisch. Man muß das Problem, das auf dem Computer gelöst werden soll, zunächst einer Analyse unterziehen. Dies ist ein sehr wichtiger Schritt, der die Qualität des späteren Programms entscheidend mitbestimmt. Damit bei diesem Schritt keine Fehler unterlaufen, bedient man sich einer Art »Checkliste«. Selbst Profi-Programmierer an einer Großrechenanlage benutzen eine ähnlich geartete Checkliste, die selbstverständlich umfangreicher ist, als die, die wir benutzen. Sie sollten sich also zu Beginn Klarheit über folgende fünf Punkte verschaffen (am besten in schriftlicher Form durch Stichpunkte oder Notizen):

»Checkliste«

1. Zweck und Funktion des Programms (Aufgabenstellung). Was soll eigentlich berechnet werden? Wie allgemein soll das Programm später anwendbar sein?
2. Welche Formeln stehen zur Lösung des Problems zur Verfügung? Sind diese Formeln genau genug? Haben sie Ausnahmen in der Gültigkeit? Können Definitionslücken auftreten?
3. Form der Dateneingabe. Welche Daten benötigt das Programm? In welcher Form werden die Daten

dem Programm zur Verfügung gestellt? (Über Tastatur, aus dem Speicher »DATA-Zeilen« oder durch Tape/Floppy oder über sonstige Wandler, »Joystick« und so weiter.) Zu welcher Genauigkeitsklasse gehören die Daten? Welcher Art sind sie? (Alphazeichen, Zahlen, alphanumerisch)
4. Anforderungen an das Programm. Welche Genauigkeit soll erzielt werden? (»=« im Zusammenhang mit späterem Variablentyp) Sollen Rundungen bei Zwischenergebnissen vorgenommen werden? (Wenn ja, auf wieviele Stellen?) Sollen Sonderfälle behandelt werden? Wenn ja, welche?
5. Datenausgabe. In welcher Form sollen die Daten ausgegeben werden? (Bildschirm, Drucker, Tape/Floppy oder über sonstige Peripherie) In welcher Form werden die Daten ausgegeben? (grafisch, als Tabelle, mit Text, im ASCII-Format)

Checkliste vor dem Start

Eine solche Problemanalyse ist ein zwingendes Muß, auch wenn Sie einfach drauflos »hacken«. Nur werden Sie sich beim »Hacken« einige dieser Fragen zwischendurch stellen, wobei Sie wichtige Punkte vergessen oder übersehen können. Sicherlich macht sich die fehlende Organisation nur zu bald bemerkbar, und so muß hier eine Änderung, dort eine Kürzung, und da noch ein Einschub gemacht werden. Das Chaos ist perfekt und die Übersicht dahin! Wenn Sie sich aber diese Fragen alle stellen, bevor Sie sich an die Tastatur begeben, können Sie sicher sein, keine wichtigen Fragen vergessen zu haben, und eine bessere Übersichtlichkeit über Ihr zukünftiges Programm zu erzielen.

Wenn Sie sich nun mit Hilfe der Checkliste einige Gedanken zu dem Programm gemacht haben, können Sie beginnen, diese Gedanken als eine Art »logischen Ablaufplan« aufzuschreiben. Dabei gehen Sie noch unabhängig von der Programmiersprache vor.

Um einen solchen Logik-Fahrplan zu konstruieren, bieten sich Ihnen zwei Darstellungsmöglichkeiten an. Zum einen das alte wohlbekannte Flußdiagramm, zum anderen die sogenannten Struktogramme, die auch nach ihrem Erfinder Nassi-Shneiderman-Diagramme genannt werden. Beide Verfahren haben ihre Vor- und Nachteile. Das Flußdiagramm besteht aus einzelnen Sym-

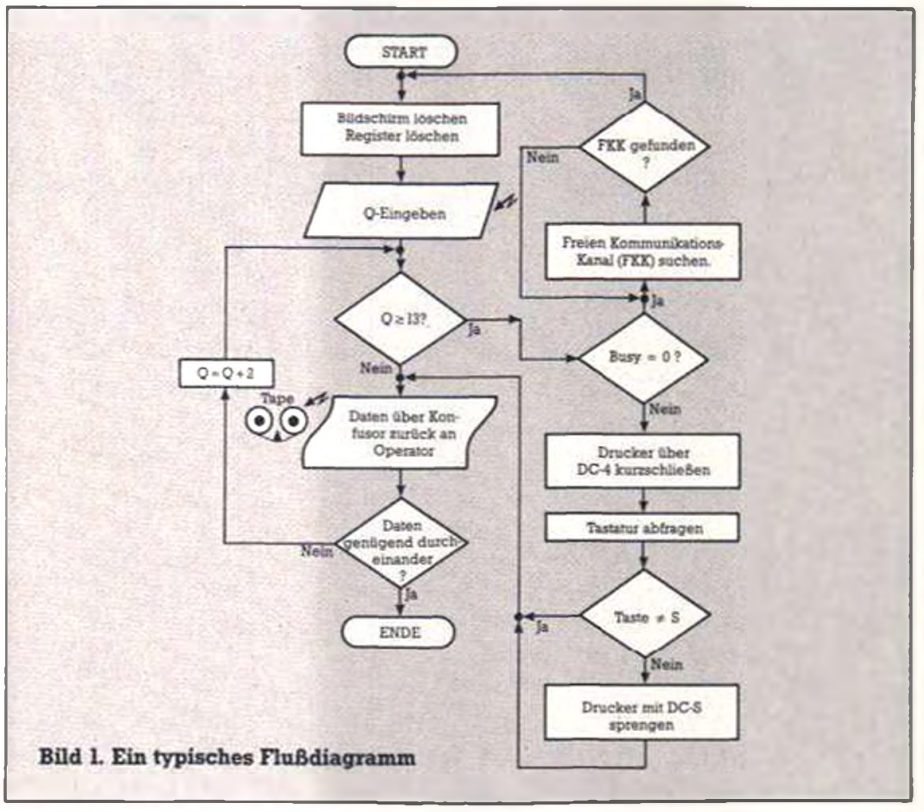


Bild 1. Ein typisches Flußdiagramm

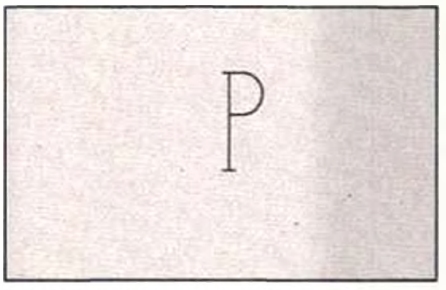


Bild 2. Das Symbol einer Sequenz

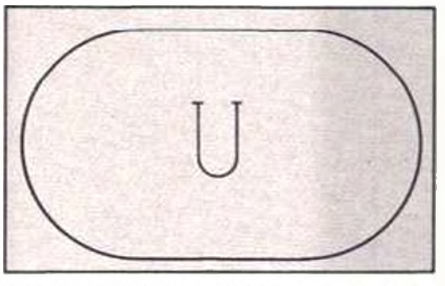


Bild 3. Aufruf eines Unterprogramms

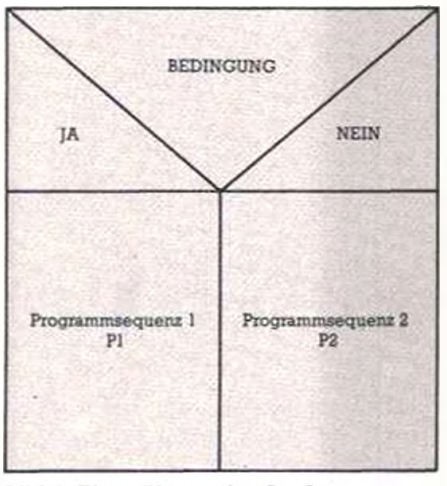


Bild 4. Eine »Alternative Struktur«

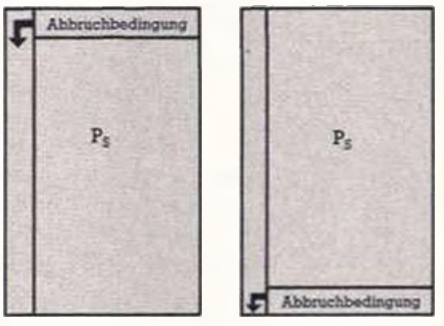


Bild 5. Symbole für Schleifenbildung

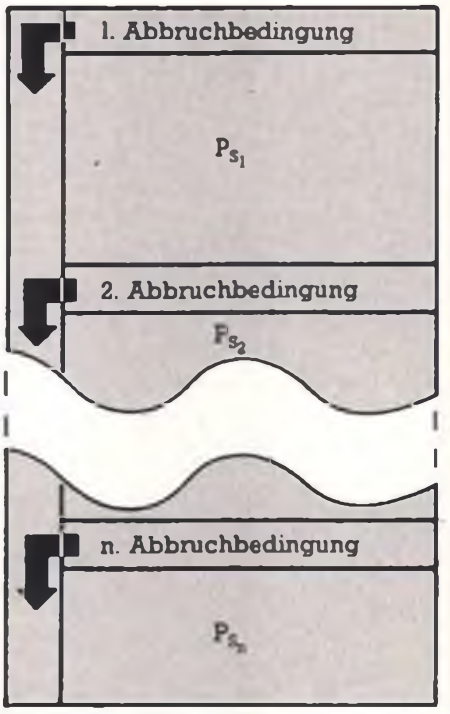


Bild 6. Beliebig viele Abbruchbedingungen sind erlaubt

bolen, die mit Pfeilen verbunden sind (siehe Bild 1). So ergibt sich eine übersichtliche Darstellung. Der Nachteil dieser Methode besteht darin, daß sie das »Spaghetti-Programmieren« mit wilden Sprüngen von einem Ende zum anderen unterstützt. Außerdem umfaßt ein Flußdiagramm viele verschiedene Symbole, die man leicht wieder vergißt. Anders bei den Nassi-Shneiderman-Diagrammen. Dort sind Sprünge gar nicht erst vorgesehen, und ein Struktogramm besteht aus nur vier Grundsymbolen. Der Nachteil liegt darin, daß man sich bei sehr langen Struktogrammen erst einlesen muß. Mit einiger Übung ist dies aber kein allzu großes Problem und so geben wir dieser Darstellungsweise den Vorzug. Um nun Struktogramme lesen und schreiben zu können, müssen wir zunächst noch kurz die vier Grundsymbole kennenlernen.

Die Sequenz:

Die Sequenz rahmt ein beliebiges Programmstück ein. In dem Kasten werden Anweisungen und Befehle im Klartext eingetragen (Bild 2).

Aufruf eines Unterprogramms:

In die Ellipse wird der Name des Unterprogramms geschrieben, welches aufgerufen wird. Das Unterprogramm selbst ist wie ein autonomes Programm dargestellt. Am Anfang des Unter-Struktogramms sollte jedoch der Name des Unterprogramms kenntlich gemacht werden (Bild 3).

Die Alternative oder IF-THEN-ELSE-Struktur:

Wird die Bedingung der Alternative erfüllt (die Aussage ist wahr = Truth), wird die linke Sequenz P1 unter JA ausgeführt, im anderen Fall (ELSE) die Sequenz P2 unter NEIN. Soll die Alternative keinen ELSE-Zweig enthalten, ist dieser durch einfaches Durchstreichen zu kennzeichnen. Man darf ihn jedoch nicht einfach weglassen (Bild 4).

Die Schleife:

Es gibt zwei Arten von Schleifen. Bei der einen steht die Abbruchbedingung am Anfang, der sogenannten WHILE-Schleife. Das bedeutet, wenn die Bedingung bereits beim Eintritt in die Schleife erfüllt ist, wird die Schleifensequenz Ps nicht durchlaufen, sondern gleich die der Schleife folgende Sequenz. Anders bei der Schleife mit der Abbruchbedingung am Ende, der sogenannten UNTIL-Schleife. Hier wird die Schleife Ps mindestens einmal durchlaufen, bevor auf die Abbruchbedingung hin geprüft wird (Bild 5).

Es können in beiden Schleifenarten beliebig viele Abbruchbedingungen auftreten (Bild 6). Ist nur eine davon erfüllt, wird die Schleife verlassen und die nachfolgende Sequenz kommt an die Reihe. Ist die Bedingung nicht erfüllt, wird in der Schleife fortgefahren, bis die nächste Abbruchbedingung vorliegt, an der erneut entschieden wird.

Diese vier Grundelemente sollten Sie sich gut einprägen. Im nächsten Teil werden wir dann das erste Struktogramm erzeugen.

Obwohl das strukturierte Programmieren fast unabhängig von der Programmiersprache ist, werden Sie feststellen, daß sich manche Dinge in bestimmten Sprachen einfacher realisieren lassen. Dennoch sollten Sie eventuelle Schwierigkeiten in »Ihrer« Sprache in Kauf nehmen. Es lohnt sich.

(Marcus Unger/ue)

Kampf dem Spaghetticode — Strukturiertes Programmieren (Teil 2)

Im zweiten Teil des Kurses lernen Sie, die Struktogramme sinnvoll einzusetzen.

Im vorangegangenen Teil haben Sie die vier Grundelemente der Nassi-Shneiderman-Diagramme, die Sequenz, den Unterprogramm-Aufruf, die Alternative und die Schleife kennengelernt. Außerdem haben wir die sogenannte Checkliste und die Programm-Grundstruktur behandelt. Aus den vier Grundelementen der Nassi-Shneiderman-Diagramme können Sie nun beginnen, einen Logik-Fahrplan (Struktogramm) zu schreiben. Dabei gehen Sie bitte wie folgt vor: Wenn Sie das erste Symbol gezeichnet haben, muß das zweite Symbol mit der gleichen Breite anschließen. Man nennt die Oberkante eines Strukturblocks auch Eingang und die Unterkante entsprechend Ausgang. Somit gilt also: Ausgang immer gleich Eingang des nachfolgenden Strukturblocks. Ein Struktogramm wird immer von oben gelesen. Wenn Sie diese Regeln beachten, ergibt sich ein (hoffentlich ernster zu nehmendes) Struktogramm ähnlich Bild 1.

Die Grundvoraussetzungen sind nun geklärt. Die Hauptregel beim strukturierten Programmieren ist, sich nicht schon am Anfang mit Details zu beschäftigen und so den Überblick über das Ganze zu verlieren. Wir überlegen also nicht schon am Anfang, was an der und der Stelle für Geräusche oder Meldungen aufzutreten haben. Es ist auch keineswegs mit einem einzigen Struktogramm getan. Für das erste »Grob-Struktogramm« werden wir unsere Checkliste und die Grundstruktur zu Hilfe nehmen. Jetzt ist es sehr wichtig, daß die Checkliste wirklich gewissenhaft beantwortet und niedergeschrieben wurde. Man beginnt zweckmäßigerweise mit der Vorbereitung zur Datenaufnahme beziehungsweise mit der Definition von Konstanten. Es wäre nun vollkommen falsch, in diesem »Grob-Stadium« »PI = 3.141592« zu schreiben. Schreiben Sie einfach »Datenaufnahme vorbereiten. Konstanten definieren«. Es ist am besten, wenn Sie immer so allgemein wie möglich bleiben und sich nicht in Details verlieren. So entwickeln Sie Schritt für Schritt das erste Struktogramm.

Im zweiten Stadium (die Verfeinerung des ersten »Grob-Stadiums«) können Sie dann schon etwas genauer beschreiben, was zu tun ist. In diesem Fall also »Variablenfelder dimensionieren. Konstante PI und XY definieren und so weiter«. Aber immer noch nicht PI = 3.141592. Dies kommt erst im letzten Schritt. Wenn Sie der Meinung sind, das Struktogramm in die Syntax Ihrer Programmiersprache übertragen zu können, ohne aus jeder Anweisung in dem Struktogramm vier oder fünf verschiedene Befehle machen zu müssen, dann ist das Struktogramm genügend verfeinert. Nun können Sie

Bild 1. Ein typisches Struktogramm

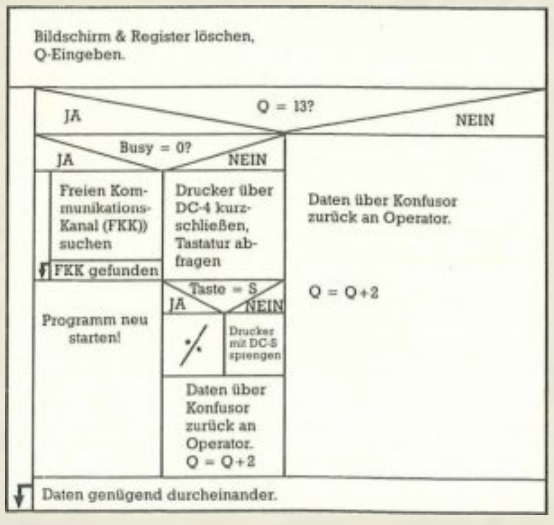
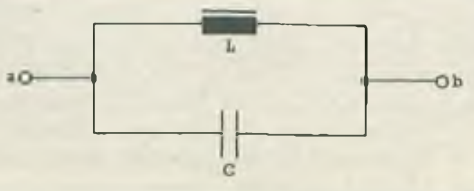


Bild 2. Schaltplan eines L-C-Schwingkreises



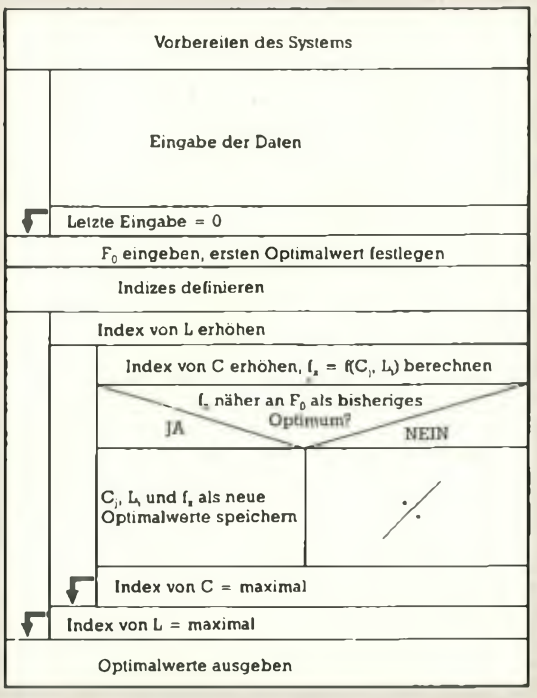


Bild 3. Grob-Struktogramm des L-C-Programms

dazu übergehen, das Struktogramm in Ihre Programmiersprache zu übersetzen. Wenn Sie Pascal oder eine ähnliche Sprache benutzen, werden Sie kaum auf Schwierigkeiten stoßen. Bei Basic hingegen können Probleme beim Übertragen der Schleifen entstehen. Falls Sie eine entsprechende Erweiterung besitzen, können Sie auch in Basic WHILE- und UNTIL-Schleifen direkt übertragen. Welche Befehle dies genau bewirken, können Sie den entsprechenden Anleitungen entnehmen. Wenn Sie jedoch nicht über eine solche Erweiterung verfügen, müssen Sie die Schleifen mit Hilfe der IF-Abfrage simulieren. Für eine While-Schleife setzen Sie die IF-Abfrage an den Anfang der Schleifensequenz. Für eine Until-Schleife aber an das Ende der zu durchlaufenden Schleifensequenz.

Der Ablauf:

a) Schriftliches Beantworten der Checkliste.

b) Aufstellen des ersten »Grob-Struktogramms« mit Hilfe der Grundstruktur und der beantworteten Checkliste.

c) 1. Verfeinerung des »Grob-Struktogramms«, 2. Verfeinerung der 1. Verfeinerung. Verfeinerung der 2. Verfeinerung.

d) Übertragen des Struktogramms in die Syntax der Programmiersprache.

e) Eintippen des Programms in den Computer.

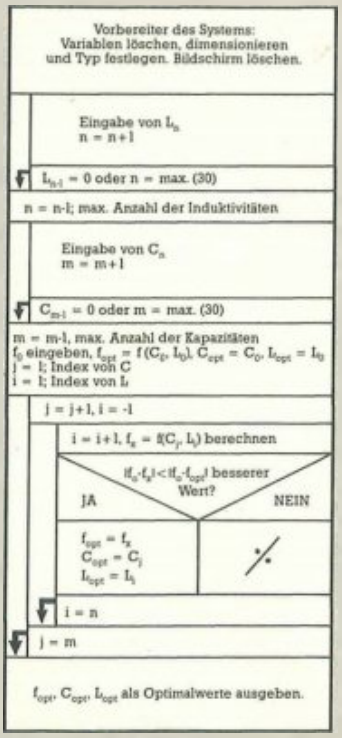


Bild 4. Verfeinerte Version des L-C-Programms

Die eigentliche Programmierarbeit endet bereits bei Punkt b). Die darauf folgenden Schritte sind eigentlich nur Routine, dürfen aber auch nicht fehlen. Wenn Sie einmal den hier beschriebenen Weg beschritten haben, und sich sicher darin fühlen, werden Sie feststellen, daß Sie nicht mehr Zeit für das Schreiben eines Programms benötigen, wie Sie sonst für das pure Hacken gebraucht hätten. Um auch allerletzten Zweiflern den Wind aus den Segeln zu nehmen, noch ein kleines Beispiel:

Stellen Sie sich folgendes Problem aus der Elektrotechnik vor: Sie wollen einen Parallelschwingkreis aus einer Induktivität (Spule) und einer Kapazität (Kondensator) aufbauen (Bild 2). Dieser parallele CL-Schwingkreis soll eine bestimmte Resonanzfrequenz F0 haben. Sie kramen aus Ihrer Bastelkiste verschiedenwertige Spulen und Kondensatoren hervor. Daß ein Bauteilpaar direkt die gewünschte Resonanz liefert, ist nicht zu erwarten. Aber welches Bauteilpaar liefert den der eigentlichen Resonanzfrequenz nächsten Wert? Das heißt: Welches Bauteilpaar verursacht den kleinsten Fehler? Angenommen Sie haben nur 20 Bauteile herausgekramt (9 Spulen und 11 Kondensatoren), dann gibt es immerhin 99 mögliche Kombinationen von Kondensator und Spule. Da ein solcher Fall häufiger eintreten kann, wird ein kleines Programm geschrieben. Selbstverständlich sollen Sie dabei »strukturiert« vorgehen. Schriftliches Beantworten der Checkliste aus Teil 1:

1. Zweck und Funktion des Programms (Aufgabenstellung):

Es soll aus einer Wertmenge mit endlich vielen Elementen und einer vorgegebenen Endbedingung das Wertepaar gefunden werden, das vom vorgegebenen Endwert am wenigsten abweicht.

2. Welche Formeln stehen für die Lösung zur Verfügung?

Die Formel für die Berechnung der Resonanzfrequenz lautet:

F0 = 1/(2xPixVLxC)

(Definitionslücke wenn L=0 oder N=0)

Die Formel gilt nur für parallele CL-Schwingkreise nach Bild 2.

3. Form der Dateneingabe:

Eingabe über Tastatur als Realzahlen. Eingaben sind mit kurzen Erläuterungen versehen, die dem Benutzer zeigen, welche Daten gefordert werden. Die Anzahl der Bauteile kann je Gruppe auf 50 beschränkt werden, da in der Praxis kaum größere Sortimente zu erwarten sind. Eine etwaige Änderung soll jedoch möglich sein.

4. Anforderung an das Programm:

Da die Werte von Bauteilen meist nur auf eine Nachkommastelle bekannt sind (Exponentialdarstellung zählt nicht als Nachkommastelle), kann keine höhere Genauigkeit erzielt werden. Es genügt also eine Genauigkeit im Rahmen der Computerstellenzahl im Zahlenbereich der reellen Zahlen. Rundungen sollen nicht vorgenommen werden. Der Sonderfall, der zur Definitionslücke in der Formel F0 führt, (C=0 oder L=0) soll schon bei der Dateneingabe abgefangen werden.

5. Ausgabe der Daten:

Die Ausgabe der Daten erfolgt über den Bildschirm in einem äquivalenten Format zur Dateneingabe. Kurze Symbole weisen den Werten ihre entsprechende Bedeutung zu.

Nachdem Sie also Ihre Checkliste beantwortet haben, gehen Sie zum nächsten Teilschritt über und zeichnen das erste Grob-Struktogramm unter Berücksichtigung der Checkliste und der in Teil 1 aufgestellten Grundstruktur (Bild 3.)

Wie vereinbart, wurden in diesem 1. Grob-Struktogramm nur ganz allgemeine und »grobe« Begriffe verwendet, die die auszuführenden Befehle kurz umreißen. Nach dem ersten Grob-Struktogramm, kann man die erste Verfeinerung in Angriff nehmen. Da das Problem nicht besonders komplex ist, reicht eine Verfeinerung (Bild 4).

Die erste Verfeinerung kann nun, ohne daß noch viel hinzugefügt werden muß, in die Syntax der verwendeten Programmiersprache übertragen werden. Wenn Sie beim Übersetzen beziehungsweise Eintippen des Programms in den Computer einige Einschübe vornehmen, werden Sie auch nach Jahren noch durch das Programm finden. Auch sollten Sie nicht zu sparsam mit Kommentaren umgehen. Wenn das Programm sehr viel Speicherplatz benötigt, können Sie notfalls mit zwei Versionen arbeiten. Die eine übersichtlich, aber nicht lauffähig. Die andere »gepackt« (ohne Leerzeichen und Kommentare), aber dafür lauffähig. So können Sie auch vorgehen, wenn Sie Ihre Programme für Dritte »unlesbar« machen wollen.

Auch wenn Ihnen zum Zeitpunkt des Programmierens die Bedeutung der Variablen vollkommen klar ist, sollten Sie stets hinter jeder Dimensionierung oder Erstbelegung einen Kommentar einfügen.

(Marcus Unger/ue)